DSA Homework 8: Spanning Trees and Shortest Paths

David Tarazi

April 10, 2020

1 Unique Edges Spanning Tree Proof

Given a graph G = (V, E) has unique edge weights then there is a single minimum spanning tree. Suppose that there are two optimal solutions, T_1 and T_2. Since the minimum spanning tree selects the minimum weight between any two nodes, then consider the first instance that T_1 and T_2 have a different edge connection. If all the weights are unique then one of the differing edges must not be the minimum edge leaving from that node, therefore contradicting the construction of a minimum spanning tree.

2 BFS Shortest Path

2.1 a

Assuming you are using a queue to keep track of which nodes to process in BFS, consider the first node, v, that is not the shortest path from the starting node, i. In breadth-first search, keeping a queue (FIFO) of the nodes means that we will process all of i's immediate neighbors before going further layers deep. Therefore, in order for v to have a path longer than the minimum, the node previous to it must have been dequeued and processed in an order that doesn't follow FIFO, creating a contradiction.

2.2 b

One possible way to create another graph, G', that is an unweighted version of G is to add a number of edges and a nodes equal to the weight between two nodes minus 1 for every edge. For instance, if there was a connection between i and j with a weight of 3, G' would create an additional two nodes and edges between i and j such that there are now 3 edges between i and j even though they are unweighted. Them, when doing BFS to find the shortest path, the way that it is processed will account for the weight between i and j such that it seems equivalent to having a weight since it essentially backlogs the queue from reaching j until the weight has expired. The new number of vertices is $(\sum w) - E + V$ and the number of edges is $\sum w$. This is because at every at edge you create the number of edges in G' equal to the weight of that edge in G. For vertices, the number of additional nodes is equal to $(\sum w) - E$ then you add on the nodes that already existed in G to find the total number of vertices.

2.3 c

The way to implement this for weighted graph G would be to use those weights to confirm when the node gets added to the processing queue. If nodes i and j had a weight of 3 between them, the algorithm should only add j to the processing queue when that weight hits 1 and the algorithm should decrease the stored weight by 1 every time that it tries to queue j. This way, j is only added at a time equal to the weight to get there. The runtime would be O(V + $\sum w$) since we have to enqueue and dequeue each vertex once, but we have to check each vertex's neighbors by the weight between them and possibly decrement the counter before we can actually add it. To find a shortest path between two nodes, we would get a runtime of O(V + $\sum w$) which is quicker than Dijkstra's algorithm but I don't actually keep track of the path itself, just the distance.